LA-UR-21-32412

Approved for public release; distribution is unlimited.

Title:	stochprop Documentation, Release 1.0
Author(s):	Blom, Philip Stephen
Intended for:	Report
Issued:	2022-02-03 (rev.1)

stochprop Documentation

Release 1.0

P. Blom

Nov 16, 2021

CONTENTS

Simulations of infrasonic propagation in the atmosphere typically utilize a single atmospheric specification describing the acoustic sound speed, ambient winds, and density as a function of altitude. Due to the dynamic and sparsely sampled nature of the atmosphere, there is a notable amount of uncertainty in the atmospheric state at a given location and time so that a more robust analysis of infrasonic propagation requires inclusion of this uncertainty. This Python library, stochprop, has been implemented using methods developed jointly by infrasound scientists at Los Alamos National Laboratory (LANL) and the University of Mississippi's National Center for Physical Acoustics (NCPA). This software library includes methods to quantify variability in the atmospheric state, identify typical seasonal variability in the atmospheric state and generate suites of representative atmospheric states during a given season, as well as perform uncertainty analysis on a specified atmospheric state given some level of uncertainty. These methods have been designed to interface between propagation modeling capabilities such as InfraGA/GeoAc and NCPAprop and signal analysis methods in the LANL InfraPy tool.

CHAPTER ONE

CONTENTS

1.1 Authorship & License Info

Authors: Philip Blom

1.2 Installation

1.2.1 Anaconda

The installation of stochprop is ideally completed using pip through Anaconda to resolve and download the correct python libraries. If you don't currently have anaconda installed on your system, please do that first. Anaconda can be downloaded from https://www.anaconda.com/distribution/.

1.2.2 Installing Dependencies

Propagation Modeling Methods

A subset of the stochprop methods require access to the LANL InfraGA/GeoAc ray tracing methods as well as the NCPAprop normal mode methods. Many of the empirical orthogonal function (EOF) based atmospheric statistics and gravity wave pertorbation methods can be used without these propagation tools, but full usage of stochprop requires them.

- InfraGA/GeoAc: https://github.com/LANL-Seismoacoustics/infraGA
- NCPAprop: https://github.com/chetzer-ncpa/ncpaprop

InfraPy Signal Analysis Methods

The propagation models constructed in stochprop are intended for use in the Bayesian Infrasonic Source Localization (BISL) and Spectral Yield Estimation (SpYE) methods in the LANL InfraPy signal analysis software suite. As with the InfraGA/GeoAc and NCPAprop linkages, many of the EOF-based atmospheric statistics methods can be utilized without InfraPy, but full usage will require installation of InfraPy (https://github.com/LANL-Seismoacoustics/infrapy).

1.2.3 Installing stochprop

Once Anaconda is installed, you can install stochprop using pip by navigating to the base directory of the package (there will be a file there named setup.py). Assuming InfraPy has been installed within a conda environment called infrapy_env, it is recommended to install stochprop in the same environment using:

```
>> conda activate infrapy_env
>> pip install -e .
```

Otherwise, a new conda environment should be created with the underlying dependencies and pip should be used to install there (work on this later):

```
>> conda env create -f stochprop_env.yml
```

If this command executes correctly and finishes without errors, it should print out instructions on how to activate and deactivate the new environment:

To activate the environment, use:

```
>> conda activate stochprop_env
```

To deactivate an active environment, use

```
>> conda deactivate
```

1.2.4 Testing stochprop

Once the installation is complete, you can test the methods by navigating to the /examples directory located in the base directory, and running:

```
>> python eof_analysis.py
>> python atmo_analysis.py
```

A set of propagation analyses are included, but require installation of infraGA/GeoAc and NCPAprop. These analysis can be run to ensure linkages are working between stochprop and the propagation libraries, but note that the simulation of propagation through even the example suite of atmosphere takes a significant amount of time.

1.3 Stochastic Propagation Analysis

- The atmospheric state at a given time and location is uncertain due to its dynamic and sparsely sampled nature
- Propagation effects for infrasonic signals must account for this uncertainty in order to properly quantify uncertainty in analysis results
- A methodology of constructing propagation statistics has been developed that identifies a suite of atmospheric states that characterize the possible space of scenarios, runs propagation simulations through each possible state, and builds statistical distributions for propagation effects

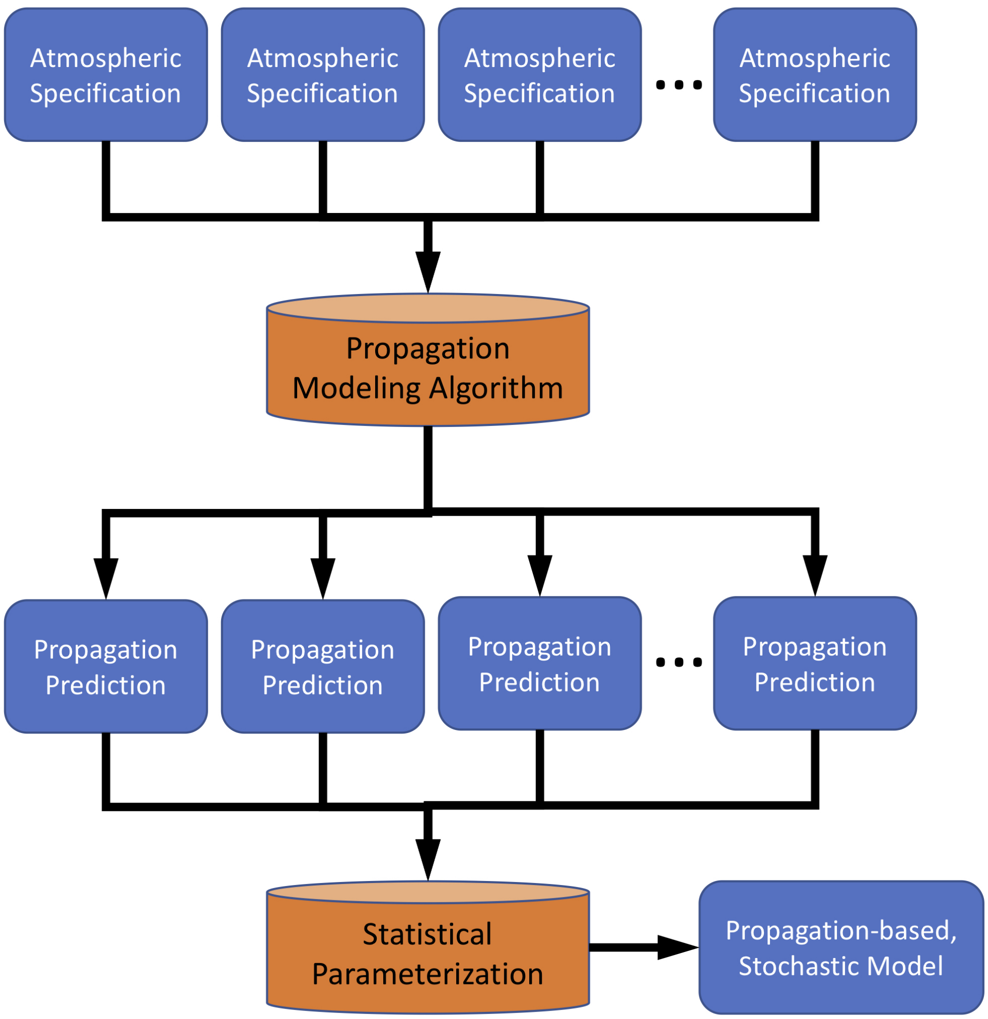


Fig. 1: Stochastic propagation models are constructing using a suite of possible atmospheric states, propagation modeling applied to each, and a statistical model describing the variability in the resulting set of predicted effects

- The tools included here provide a framework for constructing such models as well as perform a number of analyses related to atmospheric variability and uncertainty

1.3.1 Empirical Orthogonal Function Analysis

- Empirical Orthogonal Functions (EOFs) provide a mathematical means of measuring variations in the atmospheric state
- Methods measure EOF statistics to reduce the number of atmospheric samples necessary to characterize the atmosphere at a given location during a specified time period

1.3.2 Atmospheric Fitting, Sampling, and Perturbation

- EOFs can be used to fit a specified atmosphere by computing coefficients for each EOF
- Statistics of the coefficients for a suite of atmospheric states can be used to generate a set of characteristics samples
- Randomly generated EOF coefficients can be used to generate perturbations to an initial atmospheric specification and construct a suite of atmospheric states that fall within expected uncertainty

1.3.3 Propagation Statistics

- InfraGA/GeoAc ray tracing analysis can be applied to a suite of atmospheric states to predict geometric propagation characteristics such as arrival location, travel time, and direction of arrival needed to estimate the source location
- NCPAprop modal simulations can be applied to a suite of atmospheric states to predict finite frequency transmission loss needed to characterize the infrasonic source

1.3.4 Gravity Wave Perturbations

- Gravity wave perturbations are spatially and temporally sub-grid scale structures that aren't typically captured in atmospheric specifications
- The methods included here are based on the vertical ray tracing calculation discussed by Drob et al. (2013) and also investigated by Lalande & Waxler (2016)

Empirical Orthogonal Function Analysis

- Empirical orthogonal functions (EOFs) are a mathematical tool useful for characterizing a suite of vectors or functions via construction of basis vectors or functions.
- Consider N fields, $a_n(\vec{z})$, sampled at M points, z_m that define a matrix,

$$A(\vec{z}) = \begin{pmatrix} a_1(z_1) & a_2(z_1) & \cdots & a_N(z_1) \\ a_1(z_2) & a_2(z_2) & \cdots & a_N(z_2) \\ \vdots & \vdots & \ddots & \vdots \\ a_1(z_M) & a_2(z_M) & \cdots & a_N(z_M) \end{pmatrix}$$

- Analysis of this $N \times M$ matrix to compute EOFs entails first extracting the mean set of values and then applying a singular value decomposition (SVD) to define singular values and orthogonal functions,

$$A(\vec{z}) \xrightarrow{\text{SVD}} \bar{a}(z_m), \mathcal{S}_n^{(a)}, \mathcal{E}_n^{(A)}(z_m)$$

- The resulting EOF information can be used to reproduce any other other field sampled on the same set of points,

$$\hat{b}\left(z_m\right) = \bar{a}\left(z_m\right) + \sum_n \mathcal{C}_n^{(b)} \mathcal{E}_n^{(A)}\left(z_m\right),$$
$$\mathcal{C}_n^{(b)} = \sum_m \mathcal{E}_n^{(A)}\left(z_m\right)\left(b\left(z_m\right) - \bar{a}\left(z_m\right)\right),$$

- Note that the coefficients, $\mathcal{C}_n^{(b)}$, are defined by the projection of the new function onto each EOF (accounting for the mean, $\bar{a}$)
- Consider a second matrix, $B\left(\vec{z}\right)$ defined by a set of K fields, $b_k\left(\vec{z}\right)$. Each of these columns produces a set of coefficients that can be used to define a distribution via a kernel density estimate (KDE),

$$\left\{\mathcal{C}_n^{(b_1)}, \mathcal{C}_n^{(b_2)}, \ldots, \mathcal{C}_n^{(b_K)}\right\} \xrightarrow{\text{KDE}} \mathcal{P}_n^{(B)}\left(\mathcal{C}\right).$$

- Comparison of the distributions for various matrices, $B_1, B_2, B_3, \ldots$, allows one to define the relative similarity between differen sets by computing the overlap and weighting each term by the EOF singular values,

$$\Gamma_{j,k} = \sum_n \mathcal{S}_n^{(\text{all})} \int \mathcal{P}_n^{(B_j)}\left(\mathcal{C}\right) \mathcal{P}_n^{(B_k)}\left(\mathcal{C}\right) d\mathcal{C}$$

- In the case of EOF analysis for atmospheric seasonality and variability, each $a_m(\vec{z})$ is an atmospheric specification sampled at a set of altitudes, $\vec{z}$, and the set of atmospheric states in A includes all possible states for the entire year (and potentially multiple years). The sets of atmospheres in each matrix, B_j, is a subset of A corresponding to a specific month or other interval. The coefficient overlap can be computed for all combinations to identify seasonality and determine the grouping of intervals for which propagation effects will be similar.

EOF methods in stochprop

- Empirical Orthogonal Function analysis methods can be accessed by importing `stochprop.eofs`
- Although analysis can be completed using any set of user defined paths, it is recommended to build a set of directories to hold the eof results, coefficient analyses, and samples produced from seasonal analysis. It is often the case that the transitions from summer to winter and winter to summer are overly similar and can be grouped together so that only 3 season definitions are needed. This pre-analysis set up can be completed manually or by running:

```
import os
import subprocess
import numpy as np

from stochprop import eofs

if __name__ == '__main__':
        eof_dirs = ["eofs", "coeffs", "samples"]
        season_labels = ["winter", "spring", "summer"]

        for dir in eof_dirs:
                if not os.path.isdir(dir):
                        subprocess.call("mkdir " + dir, shell=True)

        for season in season_labels:
```

(continues on next page)

(continued from previous page)

```
            if not os.path.isdir("samples/" + season):
                subprocess.call("mkdir samples/" + season,␣
↪shell=True)
```

Load Atmosphere Specifications

- Atmospheric specifications are available through a number of repositories including the Ground-to-Space (G2S) system, the European Centre for Medium-Range Weather Forecasts (ECMWF), and other sources
- A convenient source for G2S specifications is the University of Mississippi's National Center for Physical Acoustics (NCPA) G2S server at http://g2s.ncpa.olemiss.edu
- The current implementation of EOF methods in stochprop assumes the ingested specifications are formatted such that the columns contain altitude, temperature, zonal winds, meridional winds, density, pressure (that is, `zTuvdp` in the infraGA/GeoAc profile options), which is the default output format of the G2S server at NCPA. Note: a script is included in the infraGA/GeoAc methods to extract profiles in this format from ECMWF netCDF files.
- The atmosphere matrix, $A(\vec{z})$ can be constructed using `stochprop.eofs.build_atmo_matrix` which accepts the path where specifications are located and a pattern to identify which files to ingest.
  - All specification in a directory can be ingested for analysis by simpy using,

```
A, z0 = eofs.build_atmo_matrix("profs/", "*.dat")
```

  - Alternately, specific months, weeks of the year, years, or hours can be defined to limit what information is included in the atmospheric matrix, $A(\vec{z})$,

```
A, z0 = eofs.build_atmo_matrix("profs/", "*.dat", months=['10',
↪'11', '12', '01', '02', '03'])
A, z0 = eofs.build_atmo_matrix("profs/", "*.dat", weeks=['01', '02
↪'])
A, z0 = eofs.build_atmo_matrix("profs/", "*.dat", years=['2010'])
A, z0 = eofs.build_atmo_matrix("profs/", "*.dat", hours=['18'])
```

Computing EOFs

- Once the atmosphere matrix, $A(\vec{z})$, has been ingested, EOF analysis can be completed using:

```
eofs.compute_eofs(A, z0, "eofs/examples")
```

- The analysis results are written into files with prefix specified in the function call ("eofs/examples" in this case). The contents of the files are summarized is the below table.

EOF Output File	Description
eofs/example-mean_atmo.dat	Mean values, $\bar{a}\,(\vec{z})$ in the above discussion
eofs/example-singular_values.dat	Singular values corresponding each EOF index
eofs/example-adiabatic_snd_spd.eofs	EOFs for the adiabatic sound speed, $c_\text{ad} = \sqrt{\gamma \frac{p}{\rho}}$
eofs/example-ideal_gas_snd_spd.eofs	EOFs for the ideal gas sound speed, $c_\text{ad} = \sqrt{\gamma RT}$
eofs/example-merid_winds.eofs	EOFs for the meridional (north/south) winds
eofs/example-zonal_winds.eofs	EOFs for the zonal (east/west) winds

- The EOF file formats is such that the first column contains the altitude points, $\vec{z}$, and each subsequent column contains the n^{th} EOF, $\mathcal{E}_n^{(A)}\,(\vec{z})$
- As discussed in Waxler et al. (2020), the EOFs are computed using stacked wind and sound speed values to conserve coupling between the different atmospheric parameters and maintain consistent units (velocity) in the EOF coefficients
- The resulting EOFs can be used for a number of analyses including atmospheric updating, seasonal studies, perturbation analysis, and similar analyses

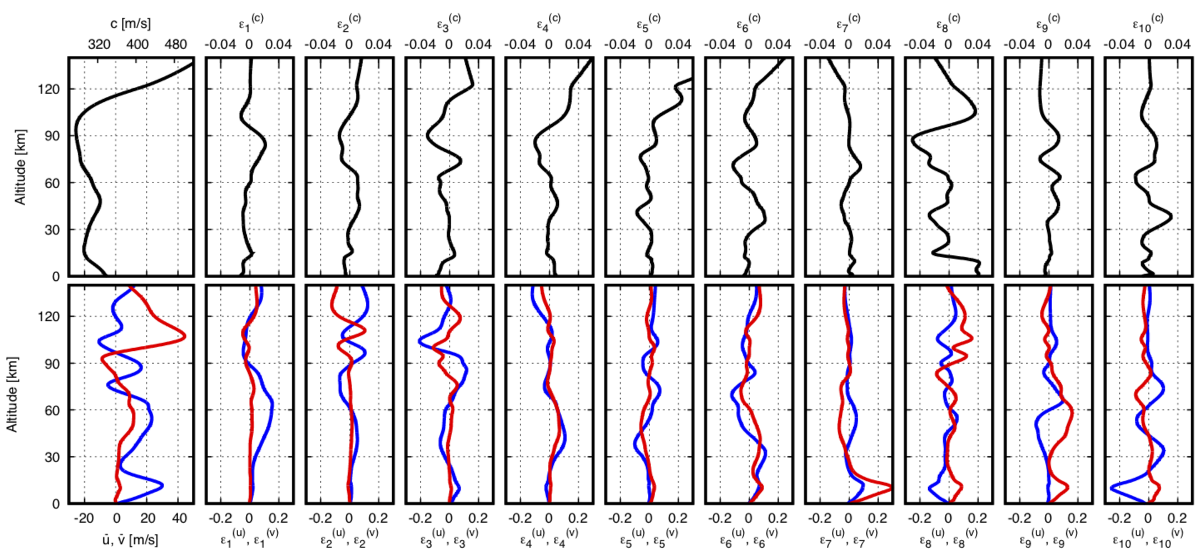


Fig. 2: Mean atmospheric states (left) and the first 10 EOFs for the adiabatic sound speed (upper row) and zonal and meridional winds (lower row, blue and red, respectively) for analysis of the atmosphere in the northeastern US

Compute Coefficients and Determine Seasonality

- Using the EOFs for the entire calendar year, coefficient sets can be defined for individual months (or other sub-intervals) using the `stochprop.eofs.compute_coeffs` function.
- For identification of seasonality by month, the coefficient sets are first computed for each individual month using:

```
coeffs = [0] * 12
for m in range(12):
    Am, zm = eofs.build_atmo_matrix("profs/", *.dat", months = ['%02d' % (m␣
↪+ 1)])
```

(continues on next page)

(continued from previous page)

```
    coeffs[m] = eofs.compute_coeffs(Am, zm, "eofs/" + run_id, "coeffs/" +␣
↪run_id + "_{:02d}".format(m + 1), eof_cnt=eof_cnt)
```

- The resulting coefficient sets are analyzed using `stochprop.eofs.compute_overlap` to identify how similar various month pairs are:

```
overlap = eofs.compute_overlap(coeffs, eof_cnt=eof_cnt)
eofs.compute_seasonality("coeffs/example-overlap.npy", "eofs/example",
↪"coeffs/example")
```

- The output of this analysis is a dendrogram identifying those months that are most similar. In the below result, May - August is identified as a consistent "summer" season, October - March as "winter", and September and April as "spring/fall" transition between the two dominant seasons

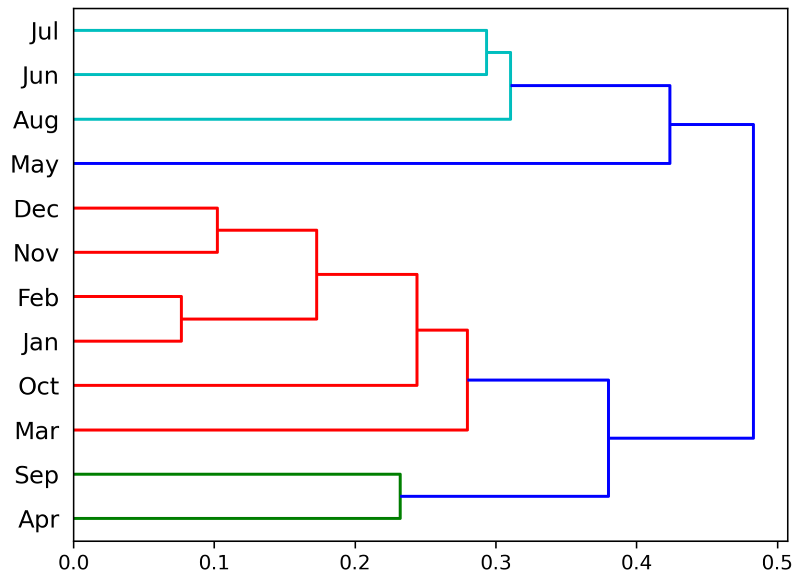


Fig. 3: Clustering analysis on coefficient overlap is used to identify which months share common atmospheric structure

Command Line interface

- A command line interface (CLI) for the EOF methods is also included and can be utilized more easily. Usage info for the EOF construction methods can be displayed by running `stochprop eof-construct --help`:

```
Usage: stochprop eof-construct [OPTIONS]

Use a SVD to construct Empirical Orthogonal Functions (EOFs) from␣
↪a suite of atmospheric specifications

Example Usage:
        stochprop eof-construct --atmo-dir profs/ --eofs-path␣
↪eofs/example
        stochprop eof-construct --atmo-dir profs/ --eofs-path␣
↪eofs/example_winter --month-selection '[10, 11, 12, 01, 02, 03]'
```

(continues on next page)

(continued from previous page)

```
Options:
  --atmo-dir TEXT          Directory of atmspheric specifications␣
↪(required)
  --eofs-path TEXT         EOF output path and prefix (required)
  --atmo-pattern TEXT      Specification file pattern (default:
↪'*.met')
  --atmo-format TEXT       Specification format (default: 'zTuvdp
↪')
  --month-selection TEXT   Limit analysis to specific month(s)␣
↪(default=None)
  --week-selection TEXT    Limit analysis to specific week(s)␣
↪(default=None)
  --year-selection TEXT    Limit analysis to specific year(s)␣
↪(default=None)
  --save-datetime BOOLEAN  Save date time info (default: False)
  --eof-cnt INTEGER        Number of EOFs to store (default: 100)
  -h, --help               Show this message and exit.
```

Atmospheric Fitting, Sampling, and Perturbation

- The Empirical Orthogonal Functions (EOFs) constructed using a suite of atmospheric specifications can be utilized in a number of different analyses of the atmospheric state
- In general, an atmospheric state can be constructed by defining a reference atmosphere, $b_0\left(z_m\right)$, and a set of coefficients, $\mathcal{C}_n$,

$$\hat{b}\left(z_m\right) = b_0\left(z_m\right) + \sum_n \mathcal{C}_n \mathcal{E}_n\left(z_m\right),$$

Fitting an Atmospheric Specification using EOFs

- In the case that a specific state, $b\left(z_m\right)$, is known, it can be approximated using the EOF set by using the mean state pulled from the original SVD analysis and coefficients defined by projecting the atmospheric state difference from this mean onto each EOF,

$$b_0\left(z_m\right) = \bar{a}\left(z_m\right), \qquad \mathcal{C}_n^{(b)} = \sum_m \mathcal{E}_n\left(z_m\right)\left(b\left(z_m\right) - \bar{a}\left(z_m\right)\right),$$

- These coefficient calculations and construction of a new atmospheric specification can be completed using `stochprop.eofs.fit_atmo` with the path to specific atmospheric state, a set of EOFs, and a specified number of coefficients to compute,

```
prof_path = "profs/01/g2stxt_2010010100_39.7393_-104.9900.dat"
eofs_path = "eofs/example"

eofs.fit_atmo(prof_path, eofs_path, "eof_fit-N=30.met", eof_cnt=30)
```

- This analysis is useful to determine how many coefficients are needed to accurately reproduce an atmospheric state from a set of EOFs. Such an analysis is shown below for varying number of coefficients and convergence is found at 50 - 60 terms.

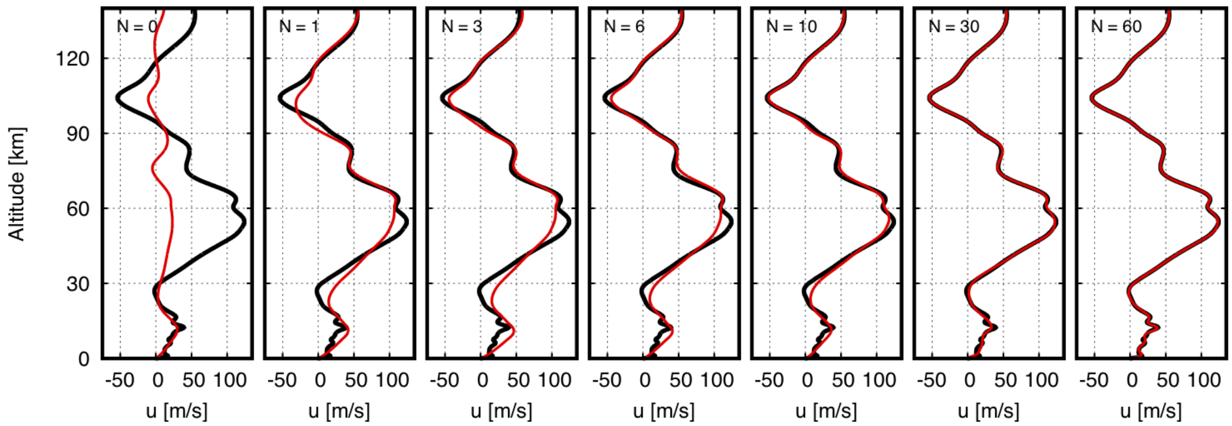


Fig. 4: Accuracy of fitting a specific atmospheric state (black) using varying numbers of EOF coefficients (red) shows convergence for approximately 50 - 60 terms in the summation

Sampling Specifications using EOF Coefficient Distributions

- Samples can be generated that are representative of a given coefficient distributions as constructed using `stochprop.eofs.compute_coeffs` or a combination of them.
- In such a case, the reference atmosphere is again the mean state from the SVD analysis and the coefficients are randomly generated from the distributions defined by kernel density estimates (KDE's) of the coefficient results

$$b_0^{(B)}\left(z_m\right) = \bar{a}\left(z_m\right), \qquad \mathcal{C}_n \longleftarrow \mathcal{P}_n^{(B)}\left(\mathcal{C}\right)$$

- In addition to sampling the coefficient distributions, the maximum likelihood atmospheric state can be defined by defining each coefficient to be the maximum of the distribution,

$$b_0^{(B)}\left(z_m\right) = \bar{a}\left(z_m\right), \qquad \mathcal{C}_n = \text{argmax}\left[\mathcal{P}_n^{(B)}\left(\mathcal{C}\right)\right]$$

- This sampling and maximum likelihood calculation can be run by loading coefficient results and running,

```
coeffs = np.load("coeffs/example_05-coeffs.npy")
coeffs = np.vstack((coeffs, np.load("coeffs/example_06-coeffs.npy")))
coeffs = np.vstack((coeffs, np.load("coeffs/example_07-coeffs.npy")))
coeffs = np.vstack((coeffs, np.load("coeffs/example_08-coeffs.npy")))

eofs.sample_atmo(coeffs, eofs_path, "samples/summer/example-summer", prof_
↪cnt=25)
eofs.maximum_likelihood_profile(coeffs, eofs_path, "samples/example-summer")
```

- This analysis can be completed for each identified season to generate a suite of atmospheric specifications representative of the season as shown in the figure below. This can often provide a significant amount of data reduction for propagation studies as multiple years of specifications (numbering in the 100's or 1,000's) can be used to construct a representative set of 10's of atmospheres that characterize the time period of interest as in the figure below.

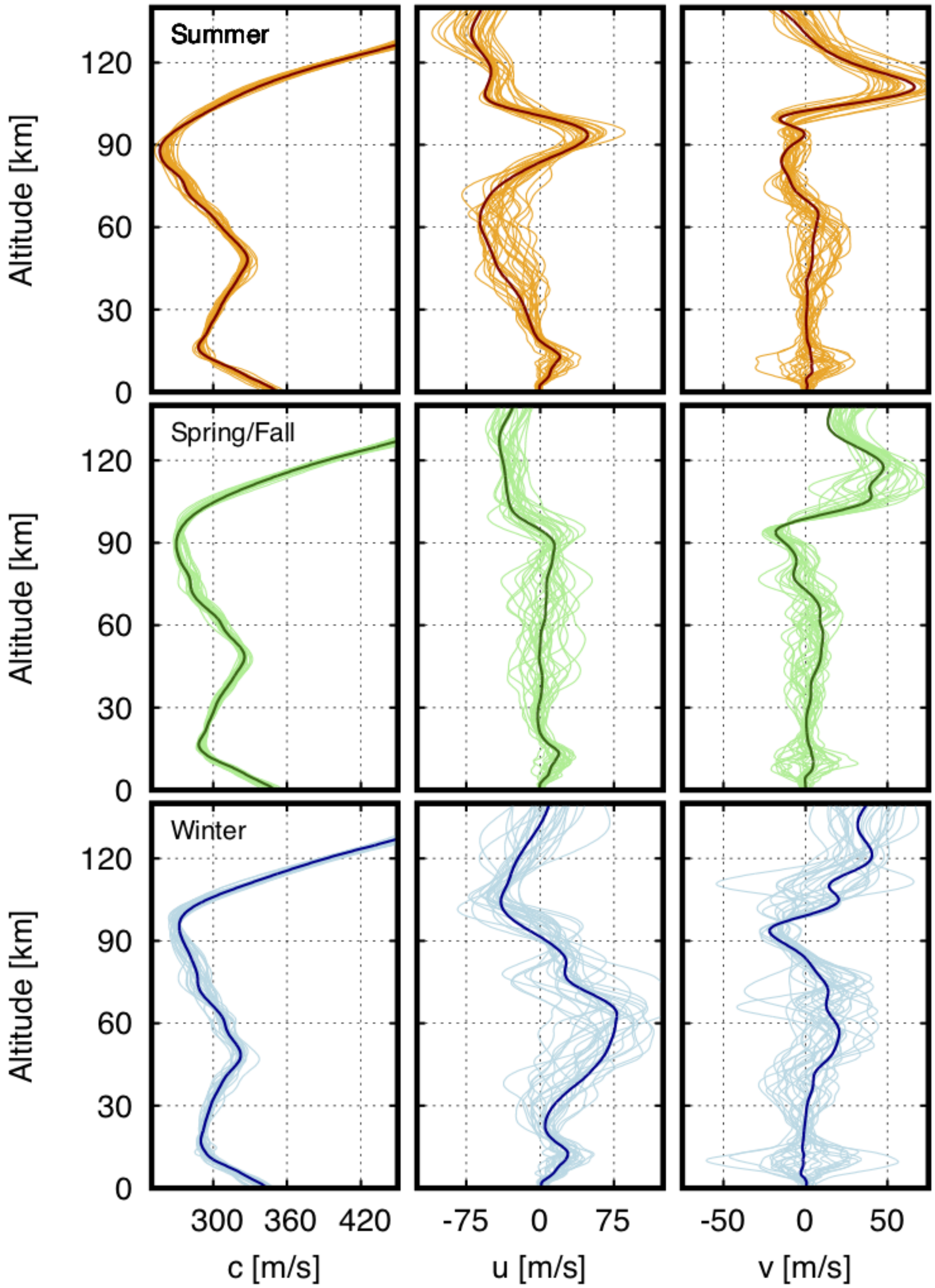


Fig. 5: Samples for seasonal trends in the western US show the change in directionality of the stratospheric waveguide in summer and winter

Perturbing Specifications to Account for Uncertainty

- In most infrasonic analysis, propagation analysis through a specification for the approximate time and location of an event doesn't produce the exact arrivals observed due to the dynamic and sparsely sampled nature of the atmosphere
- Because of this, it is useful to apply random perturbations to the estimated atmospheric state covering some confidence level and consider propagation through the entire suite of "possible" states
- In such a case, the reference atmosphere, $c_0(z_m)$ defines the initial states, coefficients are randomly generated from a normal distribution, and weighting is applied based on the singular values and mean altitudes of the EOFs,

$$b_0(z_m) = c_0(z_m), \qquad \mathcal{C}_n \longleftarrow \mathcal{N}(0, \sigma^*), \qquad w_n = \mathcal{S}_n^\gamma \, \bar{z}_n^\eta$$

- The set of perturbations is scaled to match the specified standard deviation after summing over coefficients and averaged over the entire set of altitudes
- Unlike the above methods, in this analysis a weighting is defined by the singular value of the associated EOF and the mean altitude of the EOF, $\bar{z}_n = \sum_m z_m \mathcal{E}_n(z_m)$ in order to avoid rapidly oscillating EOFs from contributing too much noise and to focus perturbations at higher altitudes where uncertainties are larger, respectively. The exponential coefficients have default values of $\gamma = 0.25$ and $\eta = 2$, but can be modified in the function call.
- This perturbation analysis can be completed using `stochprop.eofs.perturb_atmo` with a specified starting atmosphere, set of EOFs, output path, uncertainty measure in meters-per-second, and number of samples needed,

```
eofs.perturb_atmo(prof_path, eofs_path, "eof_perturb", uncertainty=5.0,␣
↪sample_cnt=10)
```

- The below figure shows a sampling of results using uncertainties of 5.0, 10.0, and 15.0 meters-per-second. The black curve is input as the estimated atmospheric state and the red curves are generated by the perturbations.

Command Line interface

- Command line methods are included to access the perturbation methods more efficiently. Usage info for the EOF perturbation methods can be displayed by running `stochprop eof-perturb --help`:

```
Usage: stochprop eof-perturb [OPTIONS]

Use a set of EOFs to perturb a reference atmospheric␣
↪specification with a defined standard deviation.

Example Usage:
        stochprop eof-perturb --atmo-file profs/g2stxt_2010010118_
↪39.7393_-104.9900.dat --eofs-path eofs/example --out test

Options:
  --atmo-file TEXT              Reference atmspheric␣
↪specification (required)
  --eofs-path TEXT              EOF output path and prefix␣
↪(required)
```

(continues on next page)

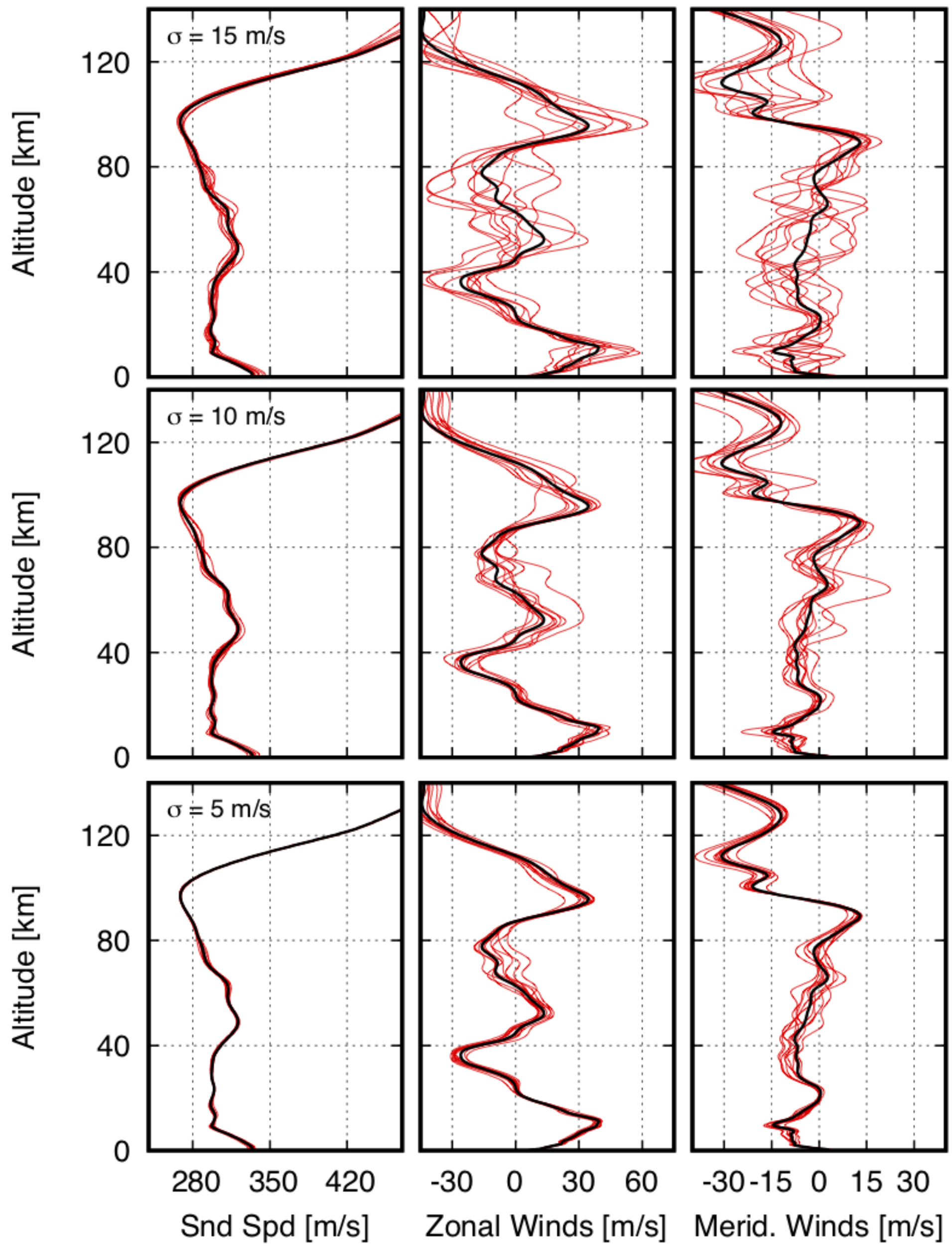


Fig. 6: Perturbations to a reference atmospheric state can be computed using randomly generated coefficients for a suite of EOFs with specified standard deviation

(continued from previous page)

```
  --out TEXT                      Output prefix (required)
  --std-dev Float                 Standard deviation (default: 10␣
↪m/s)
  --eof-max INTEGER               Maximum EOF coefficient to use␣
↪(default: 100)
  --eof-cnt INTEGER               Number of EOFs to use (default:␣
↪50)
  --sample-cnt INTEGER            Number of perturbed samples␣
↪(default: 25)
  --alt-weight FLOAT              Altitude weighting power␣
↪(default: 2.0)
  --singular-value-weight FLOAT   Sing. value weighting power␣
↪(default: 0.25)
  -h, --help                      Show this message and exit.
```

Propagation Statistics

- Propagation statistics for path geometry (e.g., arrival location, travel time, direction of arrival) and transmission loss can be computed for use in improving localization and yield estimation analyses, respectively.
- In the case of localization, a general celerity (horizontal group velocity) model is available in InfraPy constructed as a three-component Gaussian-mixture-model (GMM). This model contains peaks corresponding to the tropospheric, stratospheric, and thermospheric waveguides and has been defined by fitting the parameterized GMM to a kernel density estimate of a full year of ray tracing analyses.

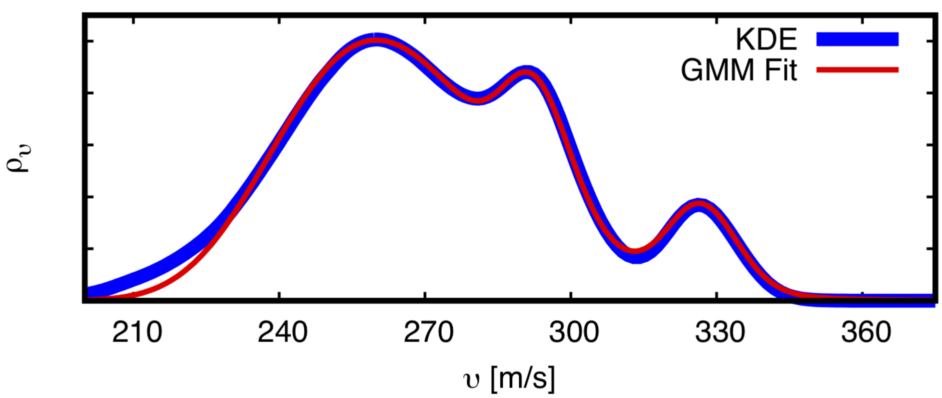


Fig. 7: A general travel time model includes three components corresponding to the tropospheric, stratospheric, and thermospheric waveguides.

- More specific models can be constructed from a limite suite of atmospheric states describing a location and seasonal trend (e.g., winter in the western US) or using an atmospheric state for a specific event with some perturbation analysis. In either case, propagation simulations are run using the suite of atmospheric states and a statistical model is defined using the outputs to quantify the probability of a given arrival characteristic.

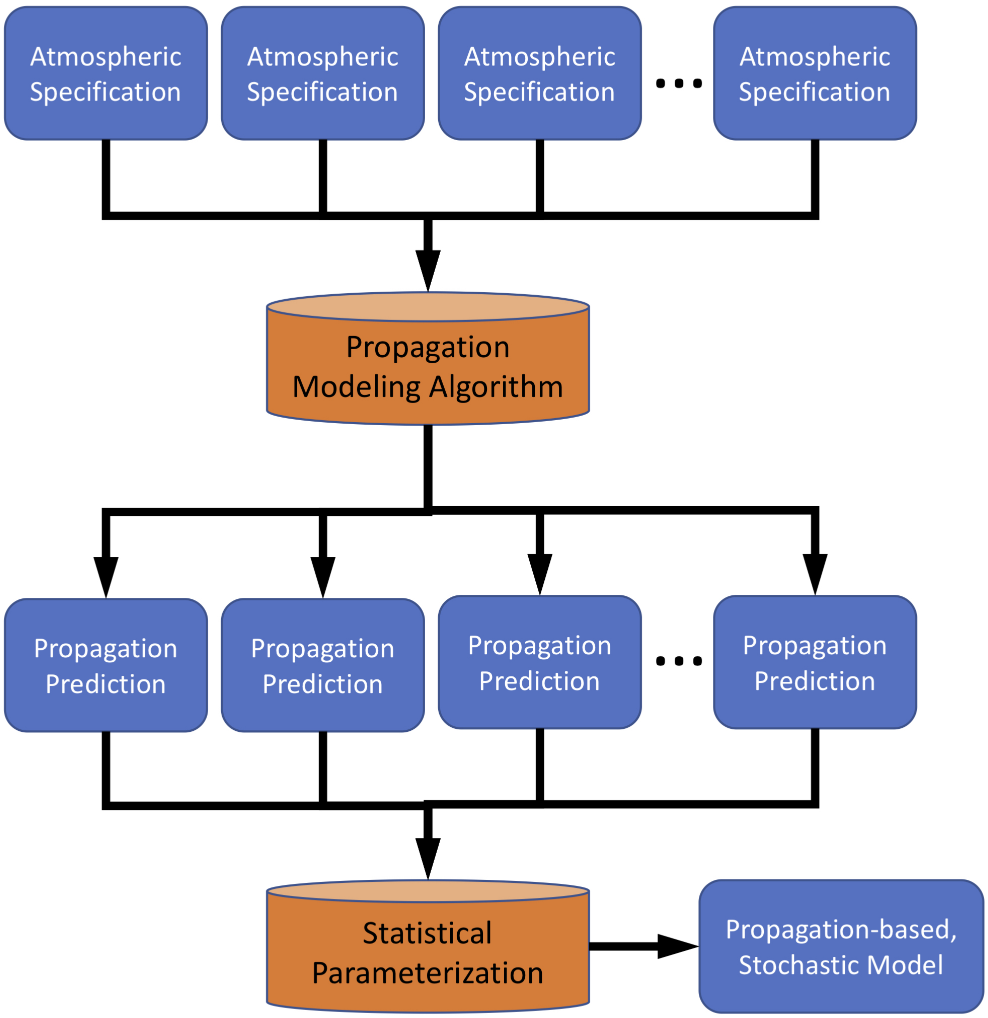


Fig. 8: Stochastic propagation models are constructing using a suite of possible atmospheric states, propagation modeling applied to each, and a statistical model describing the variability in the resulting set of predicted effects

Path Geometry Models (PGMs)

- Path geometry models describing the arrival location, travel time, direction of arrival (back azimuth, inclination angle) can be computed using geometric modeling simulations such as those in the InfraGA/GeoAc package.
- Ray tracing simulations can be run for all atmospheric specification files in a given directory using the `stochprop.propagation.run_infraga` method by specifying the directory, output file, geometry (3D Cartesian or spherical), CPU count (if the infraGA/GeoAc OpenMPI methods are installed), azimuth and inclination angle ranges, and source location
  - Note: the source location is primarily used in the spherical coordinate option to specify the latitude and longitude of the source, but should also contain the ground elevation for the simulation runs as the third element (e.g., for a source at 30 degrees latitude, 100 degrees longitude, and a ground elevation of 1 km, specify `src_loc=(0.0, 0.0, 1.0)` or `src_loc=(30.0, 100.0, 1.0)` for the `geom="3d"` or `geom="sph"` options, respectively).

```
from stochprop import propagation

propagation.run_infraga("samples/winter/example-winter", "prop/winter/
↪example-winter.arrivals.dat", cpu_cnt=12, geom="sph", inclinations=[5.0,␣
↪45.0, 1.5], azimuths=azimuths, src_loc=src_loc)
```

- The resulting infraGA/GeoAc arrival files are concatenated into a single arrivals file and can be ingested to build a path geometry model by once again specifying the geometry and source location.

```
pgm = propagation.PathGeometryModel()
pgm.build("prop/winter/example-winter.arrivals.dat", "prop/winter/example-
↪winter.pgm", geom="sph", src_loc=src_loc)
```

- The path geometry model can later be loaded into a `stochprop.propagation.PathGeometryModel` instance and visualized to investigate the propagation statistics.

```
pgm.load("prop/winter/example-winter.pgm")
pgm.display(file_id="prop/winter/example-winter", subtitle="winter")
```

- The path geometry models constructed here can be utilized in the InfraPy Bayesian Infrasonic Source Localization (BISL) analysis by specifying them as the `path_geo_model` for that analysis.

```
from infrapy.location import bisl

det_list = lklhds.json_to_detection_list('data/detection_set2.json')
result, pdf = bisl.run(det_list, path_geo_model=pgm)
```

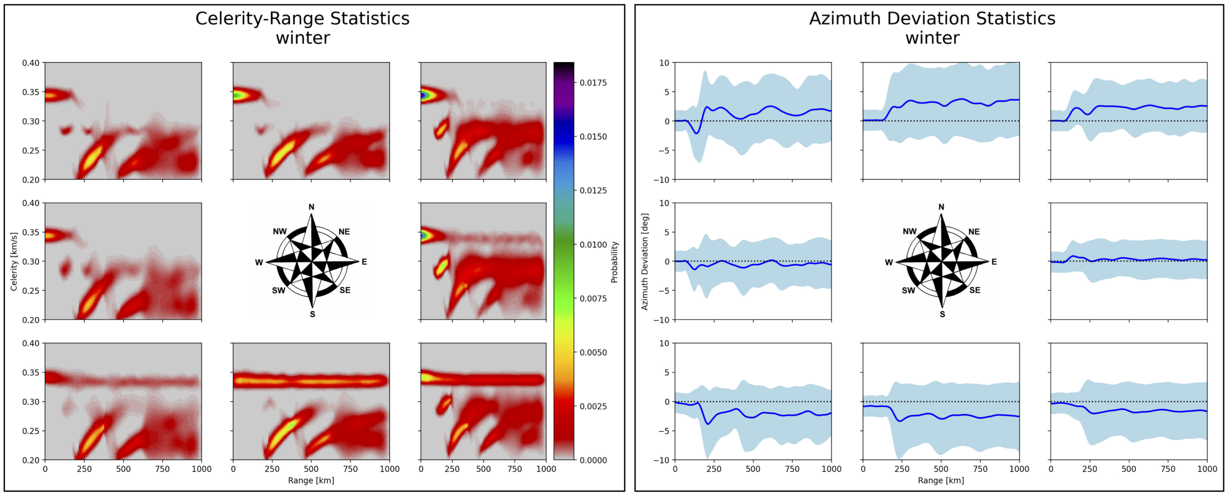


Fig. 9: Stochastic propagation-based path geometry model examples for a winter shows the expected stratospheric waveguide for propagation to the east and azimuth deviations to the north and south due to the strong stratospheric cross winds.

Transmission Loss Models (TLMs)

- Analysis of source characteristics includes estimation of the power of the acoustic signal at some reference distance from the (typically) complex source mechanism
- Such analysis using regional signals requires a propagation model that relates the energy losses along the path, termed the transmission loss and in the case of infrasonic analysis, several methods are available in the NCPAprop software suite from the University of Mississippi
- The NCPAprop modal analysis using the effective sound speed, `modess`, can be accessed from `stochprop.propagation.run_modess` to compute transmission loss predictions for all atmospheric specifications in a directory in a similar fashion to the methods above for infraGA/GeoAc.

```
from stochprop import propagation

f_min, f_max, f_cnt = 0.01, 1.0, 10
f_vals = np.logspace(np.log10(f_min), np.log10(f_max), f_cnt)

for fn in f_vals:
    propagation.run_modess("samples/winter/example-winter", "prop/winter/
↪example-winter", azimuths=azimuths, freq=fn, clean_up=True, cpu_cnt=cpu_
↪cnt)
```

- Each run of this method produces a pair of output files, `prop/winter/example-winter_0.100Hz.nm` and `prop/winter/example-winter_0.100Hz.lossless.nm` that contain the predicted transmission loss with and without thermo-viscous absorption losses.
- The transmission loss predictions are loaded in frequency by frequency and statistics for transmission as a function of propagation range and azimuth are constructed and written into specified files,

```
for fn in f_vals:
    tlm = propagation.TLossModel()
```

(continues on next page)

(continued from previous page)

```
    tlm.build("prop/winter/example-winter" + "_%.3f" %fn + ".nm", "prop/
↪winter/example-winter" + "_%.3f" %fn + ".tlm")
```

– The transmission loss model can later be loaded into a `stochprop.propagation.TLossModel` instance and visualized to investigate the propagation statistics similarly to the path geometry models.

```
tlm.load("prop/winter/example-winter_0.359Hz.tlm")
tlm.display(file_id=("prop/winter/example-winter_0.359Hz), title=(
↪"Transmission Loss Statistics" + '\n' + "winter, 0.359 Hz"))
```

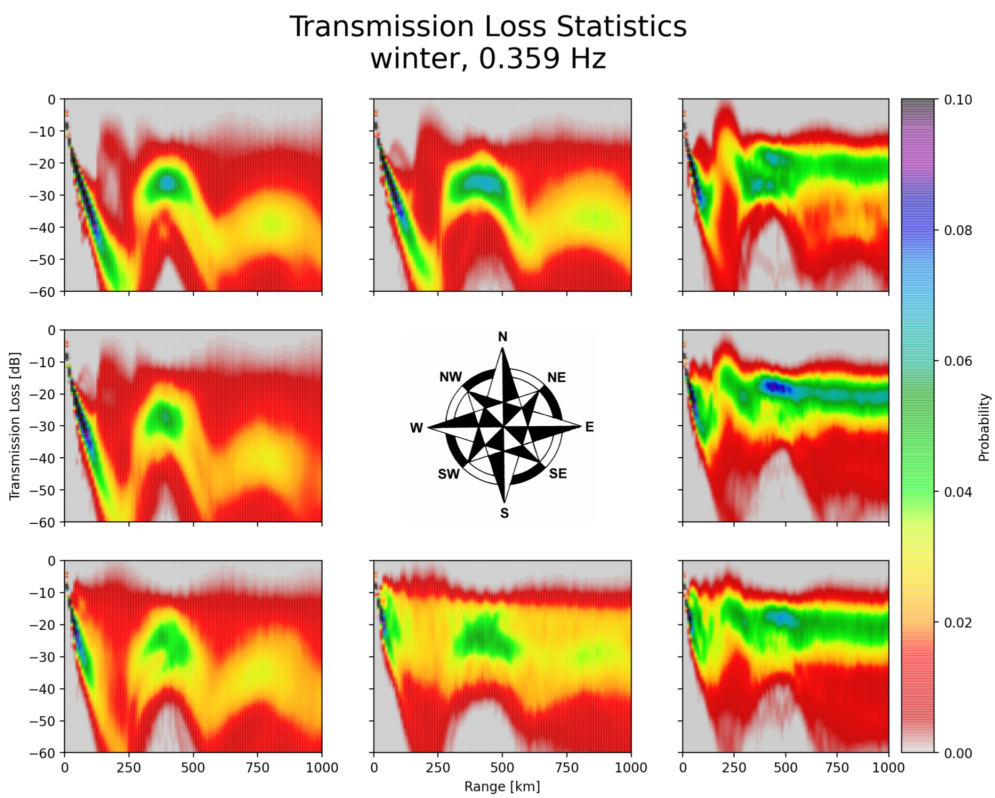


Fig. 10: Transmission loss statistics used for source characterization can be constructed using analysis of NCPAprop normal mode algorithm output.

– The transmission loss models constructed in `stochprop` can be utilized in the InfraPy Spectral Yield Estimation (SpYE) algorithm by specifying a set of models and their associated frequencies (see InfraPy example for detection and waveform data setup),

```
from infrapy.characterization import spye

# Define detection list, signal-minus-signal spectra,
# source location, and analysis frequency band

tlms = [0] * 2
tlms[0] = list(f_vals)
tlms[1] = [0] * f_cnt
```

(continues on next page)

(continued from previous page)

```
for n in range(f_cnt):
    tlms[1][n] = propagation.TLossModel()
    tlms[1][n].load("prop/winter/example-winter_" + "%.3f" % models[0][n] +
↪"Hz.tlm")

yld_vals, yld_pdf, conf_bnds = spye.run(det_list, smn_specs, src_loc, freq_
↪band, tlms)
```

Gravity Wave Perturbations

- Atmospheric specifications available for a given location and time (e.g., G2S) are averaged over some spatial and temporal scale so that sub-grid scale fluctuations must be estimated stochastically and applied in order to construct a suite of possible atmospheric states. The dominant source of such sub-grid fluctuations in the atmosphere is that of bouyancy or gravity waves.
- Stochastic gravity wave perturbation methods are included in `stochprop` using an approach based on the vertical ray tracing approach detailed in Drob et al. (2013) and are summarized below for reference.

Freely Propagation and Trapped Gravity Waves

- Gravity wave dynamics are governed by a pair relations describing the disperion and wave action conservation. The dispersion relation describing the vertical wavenumber, m, can be expressed as,

$$m^2 \left(k, l, \omega, z\right) = \frac{k_h^2}{\hat{\omega}^2} \left(N^2 - \hat{\omega}^2\right) + \frac{1}{4H^2}$$

- In this relation k and l are the zonal and meridional wave numbers, $k_h^2 = \sqrt{k^2 + l^2}$ is the combined horizontal wavenumber, $H = -\rho_0 \times \left(\frac{\partial \rho_0}{\partial z}\right)^{-1}$ is the density scale height, $\rho_0 \left(z\right)$ is the ambient atmospheric density, $N = \sqrt{-\frac{g}{\rho_0} \frac{\partial \rho_0}{\partial z}} = \sqrt{\frac{g}{H}}$ is the atmospheric bouyancy frequency, and $\hat{\omega}$ is the intrinsic angular frequency (relative to the moving air) that is defined from to the absolute angular frequency (relative to the ground), ω, horizontal wavenumbers, and winds,

$$\hat{\omega} \left(k, l, \omega, z\right) = \omega - k u_0 \left(z\right) - l v_0 \left(z\right)$$

- This dispersion relation can be solved for $\hat{\omega}$ and used to define the vertical group velocity,

$$\hat{\omega} = \frac{k_h N \left(z\right)}{\sqrt{k_h^2 + m^2 \left(z\right) + \frac{1}{4H^2(z)}}} \quad \rightarrow \quad c_{g,z} \left(k, l, \omega, z\right) = \frac{\partial \hat{\omega}}{\partial m} = -\frac{m k_h N}{\left(k_h^2 + m^2 + \frac{1}{4H^2}\right)^{\frac{3}{2}}}$$

- The conservation of wave action leads to a condition on the vertical velocity perturbation spectrum that can be used to define a freely propagating solution,

$$\rho_0 m \left|\hat{w}\right|^2 = \text{constant} \;\; \rightarrow \;\; \hat{w} \left(k, l, \omega, z\right) = \hat{w}_0 e^{i\varphi_0} \sqrt{\frac{\rho_0 \left(z_0\right)}{\rho_0 \left(z\right)} \frac{m \left(z_0\right)}{m \left(z\right)}} e^{i \int_{z_0}^z m(z') dz'}$$

- The above relation is valid in the case that $m \left(k, l, \omega, z\right)$ remains real through the integration upward in the exponential. In the case that an altitude exists for which the vertical wavenumber becomes imaginary, the gravity wave energy reflects from this turning height and the above relation is not valid. Instead, the solution is expressed in the form,

$$\hat{w}\left(k,l,\omega,z\right)=2i\sqrt{\pi}\hat{w}_0\sqrt{\frac{\rho_0\left(z_0\right)}{\rho_0\left(z\right)}\frac{m\left(z_0\right)}{m\left(z\right)}}\times\left(-r\right)^{\frac{1}{4}}\,\mathrm{Ai}\left(r\right)e^{-i\frac{\pi}{4}}S_n$$

* The Airy function argument in the above is defined uniquely above and below the turning height z_t,

$$r=\begin{cases}-\left(\frac{3}{2}\int_z^{z_t}\left|m\left(z'\right)\right|dz'\right)^{\frac{2}{3}} & z<z_t\\ \left(\frac{3}{2}\int_{z_t}^{z}\left|m\left(z'\right)\right|dz'\right)^{\frac{2}{3}} & z>z_t\end{cases}$$

* The reflection phase factor, S_n, accounts for the caustic phase shifts from the n reflections from the turning height,

$$S_n=\sum_{j=1}^{n}e^{i(j-1)\left(2\Phi-\frac{\pi}{2}\right)},\quad \Phi=\int_0^{z_t}m\left(z'\right)dz'$$

- The vertical velocity spectra defined here can be related to the horizontal velocity for the freely propagating and trapped scenarios through derivatives of the vertical velocity spectrum,

$$\hat{u}^{(\text{free})}=-\frac{km}{k_h^2}\hat{w},\quad \hat{u}^{(\text{trapped})}=\frac{2i\hat{w}_0}{\sqrt{\pi}}\frac{k}{k_h^2}\sqrt{\frac{\rho_0\left(z_0\right)}{\rho_0\left(z\right)}\frac{m\left(z_0\right)}{m\left(z\right)}}\times\left(-r\right)^{\frac{1}{4}}\,\mathrm{Ai}'\left(r\right)e^{-i\frac{\pi}{4}}S_n$$

$$\hat{v}^{(\text{free})}=-\frac{lm}{k_h^2}\hat{w},\quad \hat{v}^{(\text{trapped})}=\frac{2i\hat{w}_0}{\sqrt{\pi}}\frac{l}{k_h^2}\sqrt{\frac{\rho_0\left(z_0\right)}{\rho_0\left(z\right)}\frac{m\left(z_0\right)}{m\left(z\right)}}\times\left(-r\right)^{\frac{1}{4}}\,\mathrm{Ai}'\left(r\right)e^{-i\frac{\pi}{4}}S_n$$

- Finally, once computed for the entire atmosphere, the spatial and temporal domain forms can be computed by an inverse Fourier transform,

$$w\left(x,y,z,t\right)=\int e^{-i\omega t}\left(\iint\hat{w}\left(k,l,\omega,z\right)e^{i(kx+ly)}dk\,dl\right)d\omega$$

Damping, Source and Saturation Spectra, and Critical Layers

- At altitudes above about 100 km, gravity wave damping by molecular viscosity and thermal diffusion becomes increasingly important. Following the methods developed by Drob et al. (2013), for altitudes above 100 km, an imaginary vertical wave number term can be defined, $m\to m+m_i$, where,

$$m_i\left(k,l,\omega,z\right)=-\nu\frac{m^3}{\hat{\omega}},\quad \nu=3.563\times10^{-7}\frac{T_0^{0.69}}{\rho_0}$$

* This produces a damping factor for the freely propagating solution that is integrated upward along with the phase,

$$\hat{w}\left(k,l,\omega,z\right)=\hat{w}_0e^{i\varphi_0}\sqrt{\frac{\rho_0\left(z_0\right)}{\rho_0\left(z\right)}\frac{m\left(z_0\right)}{m\left(z\right)}}e^{i\int_{z_0}^{z}m\left(z'\right)dz'}e^{-\int_{z_0}^{z}m_i\left(z'\right)dz'}$$

* In the trapped solution, the reflection phase shift includes losses for each pass up to the turning height and back,

$$S_n=e^{-2n\Psi}\sum_{j=1}^{n}e^{i(j-1)\left(2\Phi-\frac{\pi}{2}\right)},\quad \Phi=\int_0^{z_t}m\left(z'\right)dz',\quad \Psi=\int_0^{z_t}m_i\left(z'\right)dz',$$

* Note that if z_t is below 100 km there is no loss calculated and when it is above this altitude the losses are only computed from 100 km up to the turning height.

– The source spectra defined by Warner & McIntyre (1996) specifies the wave energy density for a source at 20 km altitude (note: $\hat{\omega}$ exponential corrected in publication errata),

$$\mathcal{E}_{\text{src}}\left(m,\hat{\omega}\right) = 1.35 \times 10^{-2} \frac{m}{m_*^4 + m^4} \frac{N^2}{\hat{\omega}^{\frac{5}{3}}} \Omega, \quad \Omega = \frac{\hat{\omega}_{\text{min}}^{\frac{2}{3}}}{1 - \left(\frac{\hat{\omega}_{\text{min}}}{N}\right)^{\frac{2}{3}}}, \quad m_* = \frac{2\pi}{2.5\text{km}}$$

* The wave energy density can be expressed in terms of spectral coordiantes using $\mathcal{E}\left(k,l,\omega\right) = \mathcal{E}\left(m,\hat{\omega}\right) \frac{m}{k_h^2}$ which can then be related to the vertical velocity spectrum producing the initial condition for starting the calculation,

$$\mathcal{E}\left(k,l,\omega\right) = \frac{1}{2} \frac{N^2}{\hat{\omega}^2} \left|\hat{w}_0\right|^2 \quad \rightarrow \quad \left|\hat{w}_0\right|^2 = 2.7 \times 10^{-2} \frac{m^2}{m_*^4 + m^4} \frac{\hat{\omega}^{\frac{1}{3}}}{k_h^2} \Omega.$$

– Gravity wave breaking in the atmosphere is included in analysis via a saturation limit following work by Warner & McIntyre (1996) where the spectral coordinate saturation spectrum is (note: the exponential for $\hat{\omega}$ is again corrected in publication errata),

$$\mathcal{E}_{\text{sat}}\left(k,l,\omega\right) = 1.35 \times 10^{-2} \frac{N^2}{\hat{\omega}^{\frac{5}{3}} m^3}$$

* Again using the relation between wave energy density and vertical velocity spectrum, this produces,

$$\left|\hat{w}_{\text{sat}}\right|^2 = 2.7 \times 10^{-2} \frac{\hat{\omega}^{\frac{1}{3}}}{m^2 k_h^2}.$$

– Lastly, from the above definition for the vertical group velocity, $c_{g,z}$, it is possible to have altitudes for which $\hat{\omega} \rightarrow 0$ and $c_{g,z}$ similarly goes to zero. In such a location the wave energy density becomes infinite; however, the propagation time to such an altitude is infinite and it is therefore considered a "critical layer" because the ray path will never reach the layer. In computing gravity wave spectra using the methods here, a finite propagation time of several hours is defined and used to prevent inclusion of the critical layer effects and also quantify the number of reflections for trapped components. Drob et al. included a damping factor for altitudes with propagation times more than 3 hours and that attenuation is included here as well.

Gravity Wave implementation in stochprop

– The implementation of the gravity wave analysis partially follows that summarized by Drob et al. (2013) and is sumamrized here

* Atmospheric information is constructed from a provided atmospheric specification:

1. Interpolations of the ambient horizontal winds, $u_0\left(z\right)$ and $v_0\left(z\right)$, density, $\rho_0\left(z\right)$, and temperature, $T_0\left(z\right)$ are defined.
2. The density scale height, $H\left(z\right)$, is computed using finite differences of the ambient density.
3. Atmospheric fields are re-sampled on a higher resolution set of altitudes with $dz = 200$ meters.

* A grid of k, l, and ω values are defined:

1. The horizontal resolution, dx, is set to 4 meters following Drob et al. (2013) with $N_k = 128$ (both of these quantities can be modified by the user, but default to the values from Drob et al.)

2. Five frequency values are defined for analysis covering a frequency band from $\omega_\text{min} = 2 f_\text{Cor}$ to $\omega_\text{max} = \frac{N_\text{max}}{\sqrt{5}}$ where f_Cor is the Coriolis frequency, $f_\text{Cor} = 7.292 \times 10^{-5} \frac{\text{rad}}{\text{s}} \times \sin\left(\theta\right)$, where θ is the latitude at which the atmosphere sample was calculated.

3. Because sampling is done over intrinsic frequency, a phase shift is introduced in the Fourier transform needed to invert the solution,

$$w\left(x,y,z,t\right) = \int e^{i \hat{\omega} t} \left(\iint \hat{w}\left(k,l,\hat{\omega},z\right) e^{i\left(k u_0 + l v_0\right)} e^{i\left(kx+ly\right)} dk \, dl \right) d\hat{\omega}$$

* For each Fourier component combination, k, l, ω, several checks are made and pre-analysis completed:

  1. Those Fourier components for which $k_h > k_\text{max}$ are masked out of the calculation as well as those for which $C = \frac{N}{m} > 90 \frac{\text{m}}{\text{s}}$ and those for which $c_{g,z}\left(z_\text{src}\right) < 0.5 \frac{\text{m}}{\text{s}}$.

  2. Turning heights at which $m^2\left(z_t\right) \rightarrow 0$ are identified and for each such Fourier combination the propagation time, phase shift, and attenuation factors are computed.

* The relations above for $\hat{w}\left(k,l,\omega,z\right)$ are used to define the solution below the source height and to integrate the solution from the source height to the upper limit of the atmosphere using either the free or trapped form depending on whether a turning point exists

  1. At each altitude, the propagation time to that point is computed and compared with a user specified propagation time that defaults to 8 hours to determine whether energy has reached that altitude.

  2. Similary, the number of reflections used in computing the trapped solution phase shift if determined by the ratio of the propagation time of the trapped solution with the specified time.

  3. Unlike the Drob et al. (2013) implementation where the Fourier components are integrated upward together, the implementation in `stochprop` compute each Fourier component independently and use available `multiprocessing` tools to run the calculations in parallel. For $N_k = 128$ and $dx = 4$, the gravity wave perturbations can be computed using 10 CPUs in approximatley 20 - 30 minutes.

* The gravity wave field in the spatial and time domain are obtained by inverting the spatial components using `numpy.fft.ifft` on the appropriate axes and the ω integration is simplified by setting $t=0$ in the solution which reduces the time/frequency domain inversion to a simple integration,

$$w\left(x,y,z,0\right) = \iint \left(\int \hat{w}\left(k,l,\hat{\omega},z\right) d\hat{\omega} \right) e^{-i\left(k u_0 + l v_0\right)} e^{i\left(kx+ly\right)} dk \, dl$$

– Use of the methods is summarized in the below example:

```
from stochprop import gravity_waves

if __name__ == '__main__':
        atmo_spec = "profs/01/g2stxt_2010010100_39.7393_-104.9900.dat"
        output_path = "gw_perturb"

        t0 = 6.0 * 3600.0
```

(continues on next page)

(continued from previous page)

```
        # Run gravity wave calculation
        gravity_waves.perturb_atmo(atmo_spec, output_path, t0=t0, cpu_
↪cnt=10)
```

– A command line interface (CLI) method is also included and can be utilized more easily. General usage info can be displayed by running `stochprop gravity-waves --help`:

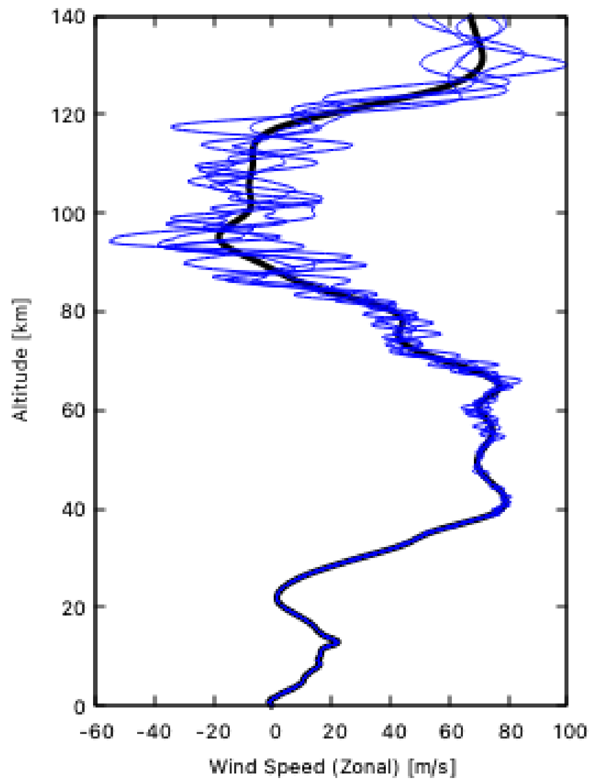
```
Usage: stochprop gravity-waves [OPTIONS]

Gravity wave perturbation calculation based on Drob et al. (2013)␣
↪method.

Example Usage:
stochprop gravity-waves --atmo-file profs/g2stxt_2010010118_39.
↪7393_-104.9900.dat --out test_gw

Options:
  --atmo-file TEXT          Reference atmspheric specification␣
↪(required)
  --out TEXT                Output prefix (required)
  --sample-cnt INTEGER      Number of perturbated samples (default:␣
↪25)
  --t0 FLOAT                Propagation time from source [hr]␣
↪(default: 8)
  --dx FLOAT                Horizontal wavenumber scale [km]␣
↪(default: 4.0)
  --dz FLOAT                Altitude resolution [km] (default: 0.2)
  --nk INTEGER              Horizontal wavenumber resolution␣
↪(default: 128)
  --nom INTEGER             Frequency resolution (default: 5)
  --random-phase BOOLEAN    Randomize phase at source [bool]␣
↪(default: False)
  --z-src FLOAT             Gravity wave source altitude [km]␣
↪(default: 20.0)
  --m-star FLOAT            Gravity wave source spectrum peak [1/
↪km] (default: (2 pi) / 2.5)
  --cpu-cnt INTEGER         Number of CPUs to use in parallel␣
↪analysis (default: None)
  -h, --help                Show this message and exit.
```

– An example CLI usage is:

```
stochprop gravity-waves --atmo-file profs/01/g2stxt_2010010100_39.7393_-104.
↪9900.dat --out gw_perturb --cpu-cnt 10
```

– An example set of perturbations is shown below.

– Note: Although perturbations to the ambient temperature are included in the Drob et al. (2013) discussion, they are not included here and modifications to the N_k, dx, and N_ω values often cause issues with the calculation of gravity waves. Work is ongoing to debug and improve the efficiency of the methods here and will be added in a future update of `stochprop`.



1.4 API

1.4.1 Empirical Orthogonal Function Analysis

stochprop.eofs.**build_atmo_matrix**(*path*, *pattern='*.dat'*, *years=None*, *months=None*, *weeks=None*, *hours=None*, *skiprows=0*, *ref_alts=None*, *prof_format='zTuvdp'*, *latlon0=None*, *return_datetime=False*)

Read in a list of atmosphere files from the path location matching a specified pattern for continued analysis.

Parameters

path: string Path to the profiles to be loaded

pattern: string Pattern defining the list of profiles in the path

skiprows: int Number of header rows in the profiles

ref_alts: 1darray Reference altitudes if comparison is needed

prof_format: string Profile format is either 'ECMWF' or column specifications (e.g., 'zTuvdp')

return_datetime: bool Option to return the datetime info of ingested atmosphere files for future reference

Returns

A: 2darray Atmosphere array of size M x (5 * N) for M atmospheres where each atmosphere samples N altitudes

z: 1darray Altitude reference values [km]

datetime: 1darray List of dates and times for each specification in the matrix (optional output, see Parameters)

`stochprop.eofs.build_cdf`(*pdf*, *lims*, *pnts=250*)

Compute the cumulative distribution of a pdf within specified limits

Parameters

pdf: function Probability distribution function (PDF) for a single variable

lims: 1darray Iterable containing lower and upper bound for integration

pnts: int Number of points to consider in defining the cumulative distribution

Returns

cfd: interp1d Interpolated results for the cdf

`stochprop.eofs.compute_coeffs`(*A*, *alts*, *eofs_path*, *output_path*, *eof_cnt=100*, *pool=None*)

Compute the EOF coefficients for a suite of atmospheres and store the coefficient values.

Parameters

A: 2darray Suite of atmosphere specifications from build_atmo_matrix

alts: 1darray Altitudes at which the atmosphere is sampled from build_atmo_matrix

eofs_path: string Path to the .eof results from compute_eofs

output_path: string Path where output will be stored

eof_cnt: int Number of EOFs to consider in computing coefficients

pool: pathos.multiprocessing.ProcessingPool Multiprocessing pool for accelerating calculations

Returns

coeffs: 2darray Array containing coefficient values of size prof_cnt by eof_cnt. Result is also written to file.

`stochprop.eofs.compute_eofs`(*A*, *alts*, *output_path*, *eof_cnt=100*)

Computes the singular value decomposition (SVD) of an atmosphere set read into an array by stochprop.eofs.build_atmo_matrix() and saves the basis functions (empirical orthogonal functions) and singular values to file

Parameters

A: 2darray Suite of atmosphere specifications from build_atmo_matrix

alts: 1darray Altitudes at which the atmosphere is sampled from build_atmo_matrix

output_path: string Path to output the SVD results

eof_cnt: int Number of basic functions to save

`stochprop.eofs.compute_overlap`(*coeffs*, *eofs_path*, *eof_cnt=100*, *method='mean'*)

Compute the overlap of EOF coefficient distributions

Parameters

coeffs: list of 2darrays

List of 2darrays containing coefficients to consider overlap in PDF of values

eofs_path: string Path to the .eof results from compute_eofs

eof_cnt: int Number of EOFs to compute

method [string] Option to decide which overlap to use ("kde" or "mean")

Returns

overlap: 3darray Array containing overlap values of size coeff_cnt by coeff_cnt by eof_cnt

`stochprop.eofs.compute_seasonality`(*overlap_file*, *file_id=None*)

Compute the overlap of EOF coefficients to identify seasonality

Parameters

overlap_file: string Path and name of file containing results of stochprop.eofs.compute_overlap

file_id: string Path and ID to save the dendrogram result of the overlap analysis

`stochprop.eofs.define_coeff_limits`(*coeff_vals*)

Compute upper and lower bounds for coefficient values

Parameters

coeff_vals: 2darrays Coefficients computed with stochprop.eofs.compute_coeffs

Returns

lims: 1darray Lower and upper bounds of coefficient value distribution

`stochprop.eofs.density`(*z*)

Computes the atmospheric density according to the US standard atmosphere model using a polynomial fit

Parameters

z: float Altitude above sea level [km]

Returns

density: float Density of the atmosphere at altitude z [g/cm^3]

`stochprop.eofs.draw_from_pdf`(*pdf*, *lims*, *cdf=None*, *size=1*)

Sample a number of values from a probability distribution function (pdf) with specified limits

Parameters

pdf: function Probability distribution function (PDF) for a single variable

lims: 1darray Iterable containing lower and upper bound for integration

cdf: function Cumulative distribution function (CDF) from stochprop.eofs.build_cfd

size: int Number of samples to generate

Returns

samples: 1darray Sampled values from the PDF

`stochprop.eofs.fit_atmo`(*prof_path*, *eofs_path*, *output_path*, *eof_cnt=100*)

Compute a given number of EOF coefficients to fit a given atmophere specification using the basic functions. Write the resulting approximated atmospheric specification to file.

Parameters

prof_path: string Path and name of the specification to be fit

eofs_path: string Path to the .eof results from compute_eofs

output_path: string Path where output will be stored

eof_cnt: int Number of EOFs to use in building approximate specification `stochprop.eofs.maximum_likelihood_profile`(*coeffs*, *eofs_path*, *output_path*, *eof_cnt=100*, *coeff_label='None'*)

Use coefficient distributions for a set of empirical orthogonal basis functions to compute the maximum likelihood specification

Parameters

coeffs: 2darrays Coefficients computed with stochprop.eofs.compute_coeffs

eofs_path: string Path to the .eof results from compute_eofs

output_path: string Path where output will be stored

eof_cnt: int Number of EOFs to use in building sampled specifications

`stochprop.eofs.perturb_atmo`(*prof_path*, *eofs_path*, *output_path*, *stdev=10.0*, *eof_max=100*, *eof_cnt=50*, *sample_cnt=1*, *alt_wt_pow=2.0*, *sing_val_wt_pow=0.25*)

Use EOFs to perturb a specified profile using a given scale

Parameters

prof_path: string Path and name of the specification to be fit

eofs_path: string Path to the .eof results from compute_eofs

output_path: string Path where output will be stored

stdev: float Standard deviation of wind speed used to scale perturbation

eof_max: int Higher numbered EOF to sample

eof_cnt: int Number of EOFs to sample in the perturbation (can be less than eof_max)

sample_cnt: int Number of perturbed atmospheric samples to generate

alt_wt_pow: float Power raising relative mean altitude value in weighting

sing_val_wt_pow: float Power raising relative singular value in weighting

`stochprop.eofs.pressure`(*z*, *T*)

Computes the atmospheric pressure according to the US standard atmosphere model using a polynomial fit assuming an ideal gas

Parameters

z: float Altitude above sea level [km]

Returns

pressure: float Pressure of the atmosphere at altitude z [mbar] and temperature T [K]

`stochprop.eofs.profiles_qc`(*path*, *pattern='*.dat'*, *skiprows=0*)

Runs a quality control (QC) check on profiles in the path matching the pattern. It can optionally plot the bad profiles. If it finds any, it makes a new direcotry in the path location called "bad_profs" and moves those profiles into the directory for you to check

Parameters

path: string Path to the profiles to be QC'd

pattern: string Pattern defining the list of profiles in the path

skiprows: int Number of header rows in the profiles

`stochprop.eofs.sample_atmo`(*coeffs*, *eofs_path*, *output_path*, *eof_cnt=100*, *prof_cnt=250*, *output_mean=False*, *coeff_label='None'*)

Generate atmosphere states using coefficient distributions for a set of empirical orthogonal basis functions

Parameters

coeffs: 2darrays Coefficients computed with stochprop.eofs.compute_coeffs

eofs_path: string Path to the .eof results from compute_eofs

output_path: string Path where output will be stored

eof_cnt: int Number of EOFs to use in building sampled specifications

prof_cnt: int Number of atmospheric specification samples to generate

output_mean: bool Flag to output the mean profile from the samples generated

1.4.2 Propagation Statistics

class stochprop.propagation.**PathGeometryModel**

Bases: `object`

Propagation path geometry statistics computed using ray tracing analysis on a suite of specifications includes celerity-range and azimuth deviation/scatter statistics

Methods

build(arrivals_file, output_file[, ...])	Construct propagation statistics from a ray tracing arrival file (concatenated from multiple runs most likely) and output a path geometry model
display([file_id, subtitle, show_colorbar])	Display the propagation geometry statistics
eval_az_dev_mn(rng, az)	Evaluate the mean back azimuth deviation at a given range and propagation azimuth
eval_az_dev_std(rng, az)	Evaluate the standard deviation of the back azimuth at a given range and propagation azimuth
eval_rcel_gmm(rng, rcel, az)	Evaluate reciprocal celerity Gaussian Mixture Model (GMM) at specified range, reciprocal celerity, and azimuth
load(model_file[, smooth])	Load a path geometry model file for use

build(*arrivals_file*, *output_file*, *show_fits=False*, *rng_width=50.0*, *rng_spacing=10.0*, *geom='3d'*, *src_loc=[0.0, 0.0, 0.0]*, *min_turning_ht=0.0*, *az_bin_cnt=16*, *az_bin_wdth=30.0*)

Construct propagation statistics from a ray tracing arrival file (concatenated from multiple runs most likely) and output a path geometry model

Parameters

arrivals_file: string Path to file containing infraGA/GeoAc arrival information

output_file: string Path to file where results will be saved

show_fits: boolean Option ot visualize model construction (for QC purposes)

rng_width: float Range bin width in kilometers

rng_spacing: float Spacing between range bins in kilometers

geom: string Geometry used in infraGA/GeoAc simulation. Options are "3d" and "sph"

src_loc: iterable [x, y, z] or [lat, lon, elev] location of the source used in infraGA/GeoAc simulations. Note: '3d' simulations assume source at origin.

min_turning_ht: float Minimum turning height used to filter out boundary layer paths if not of interest

az_bin_cnt: int Number of azimuth bins to use in analysis

az_bin_width: float Azimuth bin width in degrees for analysis

`display`(*file_id=None*, *subtitle=None*, *show_colorbar=True*)
Display the propagation geometry statistics

Parameters

file_id: string File prefix to save visualization

subtitle: string Subtitle used in figures

`eval_az_dev_mn`(*rng*, *az*)
Evaluate the mean back azimuth deviation at a given range and propagation azimuth

Parameters

rng: float Range from source

az: float Propagation azimuth (relative to North)

Returns

bias: float Predicted bias in the arrival back azimuth at specified arrival range and azimuth

`eval_az_dev_std`(*rng*, *az*)
Evaluate the standard deviation of the back azimuth at a given range and propagation azimuth

Parameters

rng: float Range from source

az: float Propagation azimuth (relative to North)

Returns

stdev: float Standard deviation of arrival back azimuths at specified range and azimuth

`eval_rcel_gmm`(*rng*, *rcel*, *az*)
Evaluate reciprocal celerity Gaussian Mixture Model (GMM) at specified range, reciprocal celerity, and azimuth

Parameters

rng: float Range from source

rcel: float Reciprocal celerity (travel time divided by propagation range)

az: float Propagation azimuth (relative to North)

Returns

pdf: float Probability of observing an infrasonic arrival with specified celerity at specified range and azimuth

`load`(*model_file*, *smooth=False*)
Load a path geometry model file for use

Parameters

model_file: string Path to PGM file constructed using stochprop.propagation.PathGeometryModel.build()

smooth: boolean Option to use scipy.signal.savgol_filter to smooth discrete GMM parameters along range

class `stochprop.propagation.TLossModel`

Bases: `object`

Methods

build(tloss_file, output_file[, show_fits, ...])	Construct propagation statistics from a NCPAprop modess or pape file (concatenated from multiple runs most likely) and output a transmission loss model
display([file_id, title, show_colorbar])	Display the transmission loss statistics
eval(rng, tloss, az)	Evaluate TLoss model at specified range, transmission loss, and azimuth
load(model_file)	Load a transmission loss file for use

build(*tloss_file*, *output_file*, *show_fits=False*, *use_coh=False*, *az_bin_cnt=16*, *az_bin_wdth=30.0*, *rng_lims=[1.0, 1000.0]*, *rng_cnt=100*, *rng_smpls='linear'*)

Construct propagation statistics from a NCPAprop modess or pape file (concatenated from multiple runs most likely) and output a transmission loss model

Parameters

tloss_file: string Path to file containing NCPAprop transmission loss information

output_file: string Path to file where results will be saved

show_fits: boolean Option ot visualize model construction (for QC purposes)

use_coh: boolean Option to use coherent transmission loss

az_bin_cnt: int Number of azimuth bins to use in analysis

az_bin_width: float Azimuth bin width in degrees for analysis

display(*file_id=None*, *title='Transmission Loss Statistics'*, *show_colorbar=True*)

Display the transmission loss statistics

Parameters

file_id: string File prefix to save visualization

subtitle: string Subtitle used in figures

eval(*rng*, *tloss*, *az*)

Evaluate TLoss model at specified range, transmission loss, and azimuth

Parameters

rng: float Range from source

tloss: float Transmission loss

az: float Propagation azimuth (relative to North)

Returns

pdf: float Probability of observing an infrasonic arrival with specified transmission loss at specified range and azimuth

load(*model_file*)

Load a transmission loss file for use

Parameters

model_file: string Path to TLoss file constructed using stochprop.propagation.TLossModel.build()

`stochprop.propagation.`**`find_azimuth_bin`**(*az*, *bin_cnt=16*)

Identify the azimuth bin index given some specified number of bins

Parameters

az: float Azimuth in degrees

bin_cnt: int Number of bins used in analysis

Returns

index: int Index of azimuth bin

`stochprop.propagation.`**`run_infraga`**(*profs_path*, *results_file*, *pattern='*.met'*, *cpu_cnt=None*, *geom='3d'*, *bounces=25*, *inclinations=[1.0, 60.0, 1.0]*, *azimuths=[- 180.0, 180.0, 3.0]*, *freq=0.1*, *z_grnd=0.0*, *rng_max=1000.0*, *src_loc=[0.0, 0.0, 0.0]*, *infraga_path=''*, *clean_up=False*)

Run the infraga -prop algorithm to compute path geometry statistics for BISL using a suite of specifications and combining results into single file

Parameters

profs_path: string Path to atmospheric specification files

results_file: string Path and name of file where results will be written

pattern: string Pattern identifying atmospheric specification within profs_path location

cpu_cnt: int Number of threads to use in OpenMPI implementation. None runs non-OpenMPI version of infraga

geom: string Defines geometry of the infraga simulations (3d" or "sph")

bounces: int Maximum number of ground reflections to consider in ray tracing

inclinations: iterable object Iterable of starting, ending, and step for ray launch inclination

azimuths: iterable object Iterable of starting, ending, and step for ray launch azimuths

freq: float Frequency to use for Sutherland Bass absorption calculation

z_grnd: float Elevation of the ground surface relative to sea level

rng_max: float Maximum propagation range for propagation paths

src_loc: iterable object The horizontal (latitude and longitude) and altitude of the source

infraga_path: string Location of infraGA executables

clean_up: boolean Flag to remove individual [..].arrival.dat files after combining

`stochprop.propagation.`**`run_modess`**(*profs_path*, *results_path*, *pattern='*.met'*, *azimuths=[- 180.0, 180.0, 3.0]*, *freq=0.1*, *z_grnd=0.0*, *rng_max=1000.0*, *ncpaprop_path=''*, *clean_up=False*, *keep_lossless=False*, *cpu_cnt=1*)

Run the NCPAprop normal mode methods to compute transmission loss values for a suite of atmospheric specifications at a set of frequency values

Note: the methods here use the ncpaprop_v2 version that includes an option for –filetag that writes output into a specific location and enables simultaneous calculations via subprocess.popen()

Parameters

profs_path: string Path to atmospheric specification files

results_file: string Path and name of file where results will be written

pattern: string Pattern identifying atmospheric specification within profs_path location

azimuths: iterable object Iterable of starting, ending, and step for propagation azimuths

freq: float Frequency for simulation

z_grnd: float Elevation of the ground surface relative to sea level

rng_max: float Maximum propagation range for propagation paths

clean_up: boolean Flag to remove individual .nm files after combining

keep_lossless: boolean Flag to keep the lossless (no absorption) results

cpu_cnt [integer] Number of CPUs to use in subprocess.popen loop for simultaneous calculations

1.4.3 Gravity Wave Perturbation Analysis

`stochprop.gravity_waves.BV_freq(H)`

Compute the Brunt-Vaisala frequency defined as :math:`N = sqrt{

**rac{g}{H}}` where** :math:`H =

rac{rho0}{ rac{partial ho_0}{partial z}` is the density scale height

Parameters

H: float Scale height, :math:`H =

ho_0 imes left(

rac{partial

ho_0}{partial z}

ight)^{-1}` Returns: f_BV: float

Brunt-Vaisalla (bouyancy) frequency, :math:`f_BV = sqrt{

rac{g}{H}}`

`stochprop.gravity_waves.cg(k, l, om_intr, H)`

Compute the vertical group velocity for gravity wave propagation as :math:`cg =

rac{partial hat{omega}}{partial m} = rac{m k_h N}{ left(k_h^2 + m^2 + rac{1}{4H^2 ight)^{ rac{3}{2}}}`

Parameters

k: float

Zonal wave number [km^{-1}]

l: float Meridional wave number [km^{-1}]

om_intr: float Intrinsic frequency (relative to winds), defined as $\hat{\omega} = \omega - ku_0 - lv_0$

H: float Scale height, :math:`H =

ho_0 imes left(

rac{partial

ho_0}{partial z}

ight)^{-1}` Returns: c_g: float

Vertical group velocity of gravity waves

`stochprop.gravity_waves.`**`m_imag`**(*k*, *l*, *om_intr*, *z*, *H*, *T0*, *d0*)

Compute the imaginary wave number component to add attenuation effects The imaginary component is defined as :math:`m_ ext{im} = -

u rac{m^3}{hat{omega}}`

where the viscosity is :math:`

u = 3.563 imes 10^{-7} rac{T_0 left(z ight)}{ ho_0 left(z ight)}`

Parameters

k: float

Zonal wave number [km^{-1}]

l: float Meridional wave number [km^{-1}]

om_intr: float Intrinsic frequency (relative to winds), defined as $\hat{\omega} = \omega - k u_0 - l v_0$

z: float Absolute height (used for turning attenuation "off" below 100 km)

H: float Scale height, :math:`H =

ho_0 imes left(

rac{partial

ho_0}{partial z}

ight)^{-1}`

T0: float Ambient temperature in the atmosphere

d0: float Ambient density in the atmosphere

Returns: m_i: float

Imaginary component of the wavenumber used for damping above 100 km (note: 100 km limit is applied elsewhere)

`stochprop.gravity_waves.`**`m_sqr`**(*k*, *l*, *om_intr*, *H*)

Compute the vertical wavenumber dispersion relation for gravity wave propagation defined as :math:`m^2 =

rac{k_h^2}{hat{omega}^2} left(N^2 - hat{omega}^2 ight) + rac{1}{4 H^2}`

Parameters

k: float

Zonal wave number [km^{-1}]

l: float Meridional wave number [km^{-1}]

om_intr: float Intrinsic frequency (relative to winds), defined as $\hat{\omega} = \omega - ku_0 - lv_0$

H: float Scale height, :math:`H =

ho_0 imes left(

rac{partial

ho_0}{partial z}

ight)^{-1}` Returns: m_sqr: float

Vertical wave number squared, :math:`m^2 =

rac{k_h^2}{hat{omega}^2 left(N^2 - hat{omega}^2

ight) +

rac{1}{4 H^2}}`

`stochprop.gravity_waves.`**`perturb_atmo`**(*atmo_spec*, *output_path*, *sample_cnt=50*, *t0=28800.0*, *dx=4.0*, *dz=0.2*, *Nk=128*, *N_om=5*, *random_phase=False*, *z_src=20.0*, *m_star=2.5132741228718345*, *env_below=True*, *cpu_cnt=None*, *fig_out=None*)

Use gravity waves to perturb a specified profile using the methods in Drob et al. (2013)

Parameters

atmo_spec: string Path and name of the specification to be used as the reference

output_path: string Path where output will be stored

sample_cnt: int Number of perturbed atmospheric samples to generate

t0: float Reference time for gravity wave propagation (typically 4 - 6 hours)

dx: float Horizontal wavenumber resolution [km]

dz: float Vertical resolution for integration steps [km]

Nk: int Horizontal wavenumber grid dimensions (Nk x Nk)

N_om: int Frequency resolution (typically 5)

ref_lat: float Reference latitude used to define the Coriolis frequency used as the minimum frequency

random_phase: boolean Controls inclusion of random initial phase shifts

env_below: boolean Controls whether perturbations below the source height are included

cpu_cnt: int Number of CPUs to use for parallel computation of Fourier components (defaults to None)

`stochprop.gravity_waves.`**`perturbations`**(*atmo_specification*, *t0=14400.0*, *dx=2.0*, *dz=0.2*, *Nk=128*, *N_om=5*, *ref_lat=40.0*, *random_phase=False*, *z_src=20.0*, *m_star=2.5132741228718345*, *figure_out=None*, *pool=None*)

Loop over Fourier components :math:`left(k, l, omega

ight)` and compute the spectral components for $\hat{u}\left(k, l, \omega, zight\right)$,

:math:`hat{v}left(k, l, omega, z

ight)`, and $\hat{w}\left(k, l, \omega, zight\right)$. Once computed, apply inverse Fourier transforms to

obtain the space and time domain forms.

Parameters

atmo_specification: string

Atmospheric specification file path

t0: float Reference time for gravity wave propagation (typically 4 - 6 hours)

dx: float Horizontal wavenumber resolution [km]

dz: float Vertical resolution for integration steps [km]

Nk: int Horizontal wavenumber grid dimensions (Nk x Nk)

N_om: int Frequency resolution (typically 5)

ref_lat: float Reference latitude used to define the Coriolis frequency used as the minimum frequency

random_phase: boolean Controls inclusion of random initial phase shifts

figure_out: string Option to output a figure with each component's structure (slows down calculations notably, useful for debugging)

pool: multiprocessing.Pool Multprocessing option for parallel computation of Fourier components

Returns

z_vals: 1darray

Altitudes of output

du_vals: 3darray Zonal (E/W) wind perturbations, du(x, y, z, t0)

dv_vals: 3darray Meridional (N/S) wind perturbations, dv(x, y, z, t0)

dw_vals: 3darray Vertical wind perturbations, dw(x, y, z, t0)

stochprop.gravity_waves.**prog_close**()

stochprop.gravity_waves.**prog_increment**(*n=1*)

stochprop.gravity_waves.**prog_prep**(*bar_length*)

stochprop.gravity_waves.**prog_set_step**(*n*, *N*, *bar_length*)

stochprop.gravity_waves.**single_fourier_component**(*k*, *l*, *om_intr*, *atmo_info*, *t0*, *src_index*, *m_star*, *om_min*, *k_max*, *random_phase=False*, *figure_out=None*, *prog_step=0*)

Compute the vertical structure of a specific Fourier component, :math:`hat{w} left(k, l, omega, z

ight), by first identifying critical layers and turning heights then using the appropriate solution form (free or trapped solution) to evalute the component.

Parameters

k: float

Zonal wave number [km^{-1}]

l: float Meridional wave number [km^{-1}]

om: float Absolute frequency (relative to the ground) [Hz]

atmo_specification: string Atmospheric specification file path

t0: float Reference time for gravity wave propagation (typically 4 - 6 hours)

src_index: int Index of the source height within the atmo_info z values

m_star: float Source parameter m_* (default value, :math:`

rac{2 pi}{2.5} ext{ km}^{-1}` is for 20 km altitude source)

om_min: float Minimum absolute frequency used in analysis

k_max: float Maximum horzintal wavenumber value used in 1 grid dimension

random_phase: bool Flag to randomize initial phase of freely propagating solution

figure_out: string Path to output figure showing component charcterisitcs

prop_step: int Progress bar increment

Returns

u_spec: 1darray

Zonal wind perturbation spectrum, hat{u}(k, l, z, omega)

v_spec: 1darray Meridional wind perturbation spectrum, hat{v}(k, l, z, omega)

w_spec: 1darray Vertical wind perturbation spectrum, hat{w}(k, l, z, omega)

eta_spec: 1darray Displacement spectrum used to compute temperature and pressure perturbations

`stochprop.gravity_waves.single_fourier_component_wrapper(args)`

1.5 References and Citing Usage

The Empirical Orthogonal Function (EOF) analyses available in `stochprop` are part of ongoing joint research between infrasound scientists at Los Alamos National Laboratory (LANL) and the University of Mississippi's National Center for Physical Acoustics (NCPA) and will be summarizing in an upcoming publication:

- Waxler, R., Blom, P., & Frazier, W. G., On the generation of statistical models for infrasound propagation from statistical models for the atmosphere: identifying seasonal and regional trends. *Geophysical Journal International*, In Preparation

Stochastic, propagation-based models for infrasonic signal analysis were initially introduced in analysis of the Bayesian Infrasonic Source Localization (BISL) and Spectral Yield Estimation (SpYE) framworks so that usage of path geometry and transmission loss models should be cited using:

- Blom, P. S., Marcillo, O., & Arrowsmith, S. J. (2015). Improved Bayesian infrasonic source localization for regional infrasound. *Geophysical Journal International*, 203(3), 1682-1693.
- Blom, P. S., Dannemann, F. K., & Marcillo, O. E. (2018). Bayesian characterization of explosive sources using infrasonic signals. *Geophysical Journal International*, 215(1), 240-251.

Gravity wave perturbation methods available here are leveraged from work by Drob et al. (2013) and Lalande & Waxler (2016) as well as supporting work referenced in those manuscripts:

- Drob, D. P., Broutman, D., Hedlin, M. A., Winslow, N. W., & Gibson, R. G. (2013). A method for specifying atmospheric gravity wavefields for long-range infrasound propagation calculations. *Journal of Geophysical Research: Atmospheres*, 118(10), 3933-3943.
- Lalande, J. M., & Waxler, R. (2016). The interaction between infrasonic waves and gravity wave perturbations: Application to observations using UTTR rocket motor fuel elimination events. *Journal of Geophysical Research: Atmospheres*, 121(10), 5585-5600.
- Warner, C. D., & McIntyre, M. E. (1996). On the propagation and dissipation of gravity wave spectra through a realistic middle atmosphere. *Journal of Atmospheric Sciences*, 53(22), 3213-3235.

See the documentation for the supporting packages (InfraGA/GeoAc, NCPAprop, InfraPy) for guidance on citing usage of those methods.

PYTHON MODULE INDEX

S

INDEX

T